String cheatsheet for coding interviews

Introduction

A string is a sequence of characters. Many tips that apply to arrays also apply to strings. You're recommended to read the page on Arrays before reading this page.
Common data structures for looking up strings:

- Trie/Prefix Tree
- Suffix Tree

Common string algorithms:

- Rabin Karp for efficient searching of substring using a rolling hash
- KMP for efficient searching of substring

Time complexity

A strings is an array of characters, so the time complexities of basic string operations will closely resemble that of array operations.

Operation	Big-O
Access	O(1)
Search	O(n)
Insert	O(n)
Remove	O(n)

Operations involving another string

Here we assume the other string is of length m.

Operation	Big-O	Note

Find substring	O(n.m)	This is the most naive case. There are more efficient algorithms for string searching such as the KMP algorithm
Concatenating strings	O(n + m)	
Slice	O(m)	
Split (by token)	O(n + m)	
Strip (remove leading and trailing whitespaces)	O(n)	

Things to look out for during interviews

Ask about input character set and case sensitivity. Usually the characters are limited to lowercase Latin characters, for example a to z.

Corner cases

- Empty string
- String with 1 or 2 characters
- String with repeated characters
- Strings with only distinct characters

Techniques

Many string questions fall into one of these buckets.

Counting characters

Often you will need to count the frequency of characters in a string. The most common way of doing that is by using a hash table/map in your language of choice. If your language has a built-in Counter class like Python, ask if you can use that instead.

If you need to keep a counter of characters, a common mistake is to say that the space complexity required for the counter is O(n). The space required for a counter of a string of latin characters is O(1) not O(n). This is because the upper bound is the range of characters, which is usually a fixed constant of 26. The input set is just lowercase Latin characters.

String of unique characters

A neat trick to count the characters in a string of unique characters is to use a 26-bit bitmask to indicate which lower case latin characters are inside the string.

```
mask = 0
for c in word:
 mask |= (1 << (ord(c) - ord('a')))
```

To determine if two strings have common characters, perform & on the two bitmasks. If the result is non-zero, ie. `mask_a & mask_b > 0`, then the two strings have common characters.

Anagram

An anagram is word switch or word play. It is the result of rearranging the letters of a word or phrase to produce a new word or phrase, while using all the original letters only once. In interviews, usually we are only bothered with words without spaces in them.

To determine if two strings are anagrams, there are a few approaches:

- Sorting both strings should produce the same resulting string. This takes O(n.log(n)) time and O(log(n)) space.
- If we map each character to a prime number and we multiply each mapped number together, anagrams should have the same multiple (prime factor decomposition). This takes O(n) time and O(1) space. Examples: Group Anagram
- Frequency counting of characters will help to determine if two strings are anagrams. This also takes O(n) time and O(1) space.

Palindrome

A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward as forward, such as `madam` or `racecar`.

Here are ways to determine if a string is a palindrome:

- Reverse the string and it should be equal to itself.
- Have two pointers at the start and end of the string. Move the pointers inward till they meet. At every point in time, the characters at both pointers should match.

The order of characters within the string matters, so hash tables are usually not helpful.

When a question is about counting the number of palindromes, a common trick is to have two pointers that move outward, away from the middle. Note that palindromes can be even or odd length. For each middle pivot position, you need to check it twice - once that includes the character and once without the character. This technique is used in Longest Palindromic Substring.

- For substrings, you can terminate early once there is no match
- For subsequences, use dynamic programming as there are overlapping subproblems. Check out this question

Essential questions

These are essential questions to practice if you're studying for this topic.

- Valid Anagram
- Valid Palindrome
- Longest Substring Without Repeating Characters

Recommended practice questions

These are recommended questions to practice after you have studied for the topic and have practiced the essential questions.

- Longest Repeating Character Replacement
- Find All Anagrams in a String
- Minimum Window Substring
- Group Anagrams
- Longest Palindromic Substring
- Encode and Decode Strings (LeetCode Premium)